**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по практической работе №4**

**по дисциплине «Программирование»**

Тема: Текстовые строки

|  |  |  |
| --- | --- | --- |
| Студент гр. |  | Полуянов В. Н. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2022

**Цель работы.**

Изучение текстовых строк как массивов символов и их обработке, а также структуры «строка» из области имен std. Изучение потоков, используемых для работы с файлами и непосредственно чтения текстовой строки из файла. Написание программы, способной отредактировать входной текст, исключив из него наиболее очевидные ошибки. Программа так же должна содержать паттерн поиска подстроки Кнута-Морриса-Пратта (далее КМП).

**Основные теоретические положения.**

Текстовые строки представляются с помощью одномерных массивов символов. В изучаемом языке C++ текстовая строка представляет собой набор символов, заканчивающийся нулевым символом ‘\n’. Поэтому для текстовой строки необходимо выделить память под N+1 символов, где N – количество символов в строке.

Чтение текстовых строк из потока ввода осуществляется до первого пробельного символа. Таким образом, чтобы прочесть всю введённую строку полностью, следует воспользоваться функцией получения строки из потока ввода.

Основными задачами при обработке текстовых строк являются следующие: определение фактической длины текста, копирование текста из одной строки в другую, объединение строк, сравнение (больше, меньше, равно) строк и т.п.

Для удобной работы с строками существует высокоуровневый класс string в пространстве имен std. Этот класс предоставляет готовый набор методов для работы со строками, что значительно упрощает разработку и уменьшает объем исходного кода, делая его более читабельным.

**Постановка задачи.**

Разработать алгоритм и написать программу, которая:

1. Считывает строку с файла или с клавиатуры. Способ ввода выбирается пользователем.
2. Редактирует входной текст
   1. Удаляет лишние пробелы
   2. Удаляет лишние знаки препинания
   3. Исправляет регистр букв
3. Выводит на экран все слова, содержащие
   1. Только буквы
   2. Только цифры
   3. И буквы, и цифры
4. Выводит на экран ту же строку, заменив во всех словах цифры на буквы латинского алфавита, номера которых соответствуют заменяемой цифре (0 – a, 1 – b и т.д.)
5. Осуществляет поиск подстроки, введённой с клавиатуры.
   1. Производит поиск с помощью алгоритма линейного поиска
   2. Производит поиск с помощью алгоритма КМП

**Выполнение работы.**

Код программы представлен в приложении А.

1. При запуске программы пользователю предлагается ввести строку. Если пользователь оставит ввод пустым, то строка будет прочитана из файла (рис. 1).



Рисунок 1. Запуск программы и ввод строки

1. После ввода пользователю выводится подсказка команды, которая выводит меню доступных функций и ожидается ввод команды с клавиатуры (рис. 2).



Рисунок 2. Главное меню

1. Следующий шаг зависит от введенной команды, если пользователь ввёл:
   1. “0”, то выполнение программы завершается.
   2. “1”, то текст редактируется: удаляются лишние пробелы и знаки препинания, исправляется регистр букв (рис. 3)

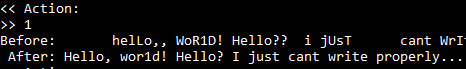


Рисунок 3. Задача 1

* 1. “2”, то поочередно выводятся: все слова, содержащие только буквы, только цифры и слова, содержащие и буквы, и слова (рис. 4).

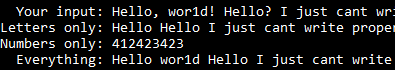


Рисунок 4. Задача 2

* 1. “3”, то выполняется замена всех встречаемых цифр в строке на соответствующие им буквы («соответствующие» - имеющие номер в латинском алфавите, равный заменяемой цифру) (рис. 5).

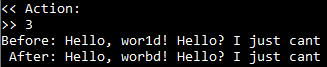


Рисунок 5. Задача 3

* 1. “4”, то выполняется поиск всех вхождений подстроки линейным алгоритмом и паттерном КМП (рис. 6).

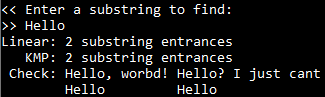


Рисунок 6. Задача 4

* 1. “h”, то выводится меню доступных команд (рис. 7).

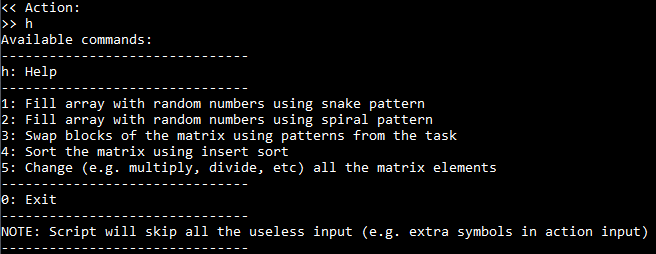


Рисунок 7. Меню

* 1. Любой символ или набор символов, отличный от существующий команд, то будет выведено сообщение об ошибке (рис. 8).



Рисунок 8. Неизвестная команда

**Выводы.**

В ходе работы были изучены текстовые строки как массивы, класс string и методы обработки строк. Изучен паттерн поиска подстроки в строке Кнута-Морриса-Пратта.

Приложение А

рабочий код

#include <iostream>  
#include <iomanip>  
#include <fstream>  
  
unsigned short getInput(std::string &str) {  
 // Default input  
 std::cout << "<< Write the sentence (or just press the Enter to apply reading from the input.txt):\n>> ";  
 std::cin.sync();  
 std::getline(std::cin, str);  
  
 // Read from file if input is empty  
 if (str.empty()) {  
 const char\* filePath = "C:/dev/university/cpp/practice\_4/input.txt";  
 std::ifstream file(filePath, std::ios::in);  
 if (!file.is\_open()) {  
 std::cout << "FileNotFoundError: No such file or directory\n";  
 return 1;  
 }  
 while (!file.eof()) std::getline(file, str);  
 file.close();  
 }  
 return 0;  
}  
  
void getAlphabet(std::string &str, bool withCapital = false, bool withLower = false, bool withNumbers = false) {  
 if (!withCapital && !withLower && !withNumbers) return;  
  
 std::string upper = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";  
 std::string lower = "abcdefghijklmnopqrstuvwxyz";  
 std::string numbers = "0123456789";  
  
 if (withCapital) str += upper;  
 if (withLower) str += lower;  
 if (withNumbers) str += numbers;  
}  
  
int findCharEntrance(char chr, const std::string &str) {  
 for (int i = 0; i < str.length(); ++i)  
 if (str[i] == chr) return i;  
 return -1;  
}  
  
void removeExtraSpaces(std::string &str) {  
 unsigned long length = str.length();  
  
 // Left trim  
 for (int i = 0; i < length; ++i) {  
 if (str[i] != ' ') break;  
 str = str.substr(i+1, --length);  
 }  
  
 // Trim spaces between words  
 for (int i = 0; i < length; ++i) {  
 if (str[i] == ' ' && str[i+1] == ' ') {  
 str = str.substr(0, i) + str.substr(i+1, --length);  
 --i;  
 }  
 }  
  
 // Right trim  
 if (str[length-1] == ' ') str = str.substr(0, length-2);  
}  
  
void correctPunctuation(std::string &str) {  
 unsigned long length = str.length();  
  
 for (int i = 0; i < length; ++i) {  
 // \W[. ! ? !? ... !!! ???]\s[A-Z0-9]  
 if (findCharEntrance(str[i], ".!?") != -1) {  
 // Wrong before the symbol  
 if (findCharEntrance(str[i-1], " .?!,([{+-\*") != -1) {  
 str = str.substr(0, i-1) + str.substr(i, length--);  
 i--;  
 }  
 if (str[i + 1] == ' ') continue; // Space after symbol - all is correct  
 if ((str[i] == '!' && str[i+1] == '?') || (str[i] == '?' && str[i+1] == '!')) i++; // ?! or !?  
 if (str[i] == str[i+1] && str[i+1] == str[i+2]) i += 2; // Triple  
 for (int j = i+1; findCharEntrance(str[j], ".?!,()[]{}+-\*") != -1; ++j) // Another symbols  
 str = str.substr(0, j) + str.substr(j + 1, length--);  
 }  
  
 // \W[,)]}]\s\W || \W[([{+-\*]\W  
 if (findCharEntrance(str[i], ",+-\*()[]{}") != -1) {  
 // Space before  
 if (str[i-1] == ' ') {  
 str = str.substr(0, i-1) + str.substr(i, length--);  
 i--;  
 }  
 // Another symbol before the current  
 if (findCharEntrance(str[i-1], " .?!,([{+-\*") != -1) {  
 str = str.substr(0, i) + str.substr(i+1, length--);  
 i--;  
 }  
 // Space afterwards  
 if (str[i+1] == ' ' && findCharEntrance(str[i+1], "([{+-\*") != -1)  
 str = str.substr(0, i-1) + str.substr(i, length--);  
 // Another symbols  
 for (int j = i+1; findCharEntrance(str[j], ".?!,()[]{}+-\*") != -1; ++j)  
 str = str.substr(0, j) + str.substr(j + 1, length--);  
 }  
 }  
}  
  
void correctLetterCase(std::string &str) {  
 unsigned long length = str.length();  
 std::string alphabetLower, alphabetUpper;  
 getAlphabet(alphabetLower, false, true, false);  
 getAlphabet(alphabetUpper, true, false, false);  
  
 // First letter always capital  
 if (findCharEntrance(str[0], alphabetUpper) == -1)  
 str[0] = alphabetUpper[findCharEntrance(str[0], alphabetLower)];  
  
 // All not after .?! is lower  
 for (int i = 1; i < length; ++i) {  
 // Lower  
 if (findCharEntrance(str[i], alphabetUpper) != -1)  
 str[i] = alphabetLower[findCharEntrance(str[i], alphabetUpper)];  
  
 // New sentence  
 if (findCharEntrance(str[i-2], ".?!") != -1  
 && findCharEntrance(str[i], alphabetLower) != -1) {  
 str[i] = alphabetUpper[findCharEntrance(str[i], alphabetLower)];  
 }  
 }  
}  
  
void showFilteredByStr(std::string &str, bool withCapital = false, bool withLower = false, bool withNumbers = false) {  
 unsigned long length = str.length();  
 std::string fullAlphabet = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";  
 std::string alphabet;  
 getAlphabet(alphabet, withCapital, withLower, withNumbers);  
  
 // Split to words  
 std::string words[100];  
 std::string current;  
 unsigned short counter = 0;  
 for (int i = 0; i < length; ++i) {  
 if (findCharEntrance(str[i], fullAlphabet) != -1) {  
 current += str[i];  
 } else if (!current.empty()){  
 words[counter++] = current;  
 current = "";  
 }  
 }  
 words[counter++] = current;  
  
 // Filter  
 for (int i = 0; i < counter; ++i) {  
 unsigned int wordLen = words[i].size()/sizeof(char);  
 bool isWordCorrect = true;  
 for (int j = 0; j < wordLen; ++j) {  
 if (findCharEntrance(words[i][j], alphabet) == -1) {  
 isWordCorrect = false;  
 break;  
 }  
 }  
 if (isWordCorrect) std::cout << words[i] << ' ';  
 }  
 std::cout << '\n';  
}  
  
void replaceIntsWithChars(std::string &str) {  
 std::string numbers;  
 getAlphabet(numbers, false, false, true);  
 for (char & chr : str)  
 if (findCharEntrance(chr, numbers) != -1) chr += 49;  
}  
  
int findAllLinear(std::string &subStr, std::string &str, int \*result) {  
 int count = 0;  
 for (int i = 0; str[i]; ++i)  
 for (int j = 0; str[i+j] == subStr[j]; ++j)  
 if (j == subStr.length() - 1) result[count++] = i;  
 return count;  
}  
  
int findAllFast(std::string &subStr, std::string &str, int \*result) {  
 if (subStr.length() > str.length()) return -1;  
  
 // Init  
 unsigned long prefixesLength = subStr.length() + 1 + str.length();  
 int prefixes[prefixesLength];  
 std::string concatStr = subStr + std::string("$") + str;  
 unsigned long length = concatStr.length();  
  
 // KMP  
 prefixes[0] = 0;  
 int count = 0;  
 for (int i = 1; i < length; ++i) {  
 int j = prefixes[i-1];  
 while ((j > 0) && (concatStr[i] != concatStr[j]))  
 j = prefixes[j-1];  
  
 if (concatStr[i] == concatStr[j])  
 ++j;  
 prefixes[i] = j;  
 if (j == subStr.length()) result[count++] = i - 2\*j;  
 }  
 return count;  
}  
  
int main() {  
 // Init the string  
 std::string input;  
 unsigned short response = getInput(input);  
 if (response == 1) return 1;  
  
 // Main loop  
 std::cout << "Enter 'h' to get list of commands\n";  
 while (true) {  
  
 // Get command from user  
 char userAction;  
 std::cout << "<< Action:\n>> ";  
 std::cin.sync();  
 std::cin >> userAction;  
  
 // Exit  
 if (userAction == '0') break;  
  
 switch (userAction) {  
  
 // Edit text  
 case '1': {  
 std::cout << "Before: " << input << std::endl;  
 removeExtraSpaces(input);  
 correctPunctuation(input);  
 correctLetterCase(input);  
 std::cout << " After: " << input << std::endl;  
 break;  
 }  
  
 // Filter words by letters or numbers presence  
 case '2': {  
 std::cout << " Your input: " << input << std::endl;  
 std::cout << "Letters only: ";  
 showFilteredByStr(input, true, true, false);  
 std::cout << "Numbers only: ";  
 showFilteredByStr(input, false, false, true);  
 std::cout << " Everything: ";  
 showFilteredByStr(input, true, true, true);  
 break;  
 }  
  
 // Replace numbers to letters with the same index in the alphabet as number value  
 case '3': {  
 std::cout << "Before: " << input << std::endl;  
 replaceIntsWithChars(input);  
 std::cout << " After: " << input << std::endl;  
 break;  
 }  
  
 // Find substring in string via linear and KMP algorithms  
 case '4': {  
 std::cout << "<< Enter a substring to find:\n>> ";  
 std::string toFind;  
 std::cin.sync();  
 std::getline(std::cin, toFind);  
  
 // Get the answer  
 int entrances[10];  
 int count = findAllLinear(toFind, input, entrances);  
 std::cout << "Linear: " << count << " substring entrances" << std::endl;  
 count = findAllFast(toFind, input, entrances);  
 std::cout << " KMP: " << count << " substring entrances" << std::endl;  
  
 // Check the answer correctness  
 std::cout << " Check: " << input << '\n';  
 for (int i = 0; i < 8; ++i) std::cout << ' ';  
 for (int i = 0; i < count; ++i) {  
 int spacesN = (i == 0) ? entrances[i] : entrances[i] - entrances[i-1] - int(toFind.length());  
 for (int j = 0; j < spacesN; ++j) std::cout << ' ';  
 std::cout << toFind;  
 }  
 std::cout << std::endl;  
  
 break;  
 }  
  
 // Update the user input  
 case 'c': {  
 response = getInput(input);  
 if (response == 1) return 1;  
 break;  
 }  
  
 // Help  
 case 'h': {  
 std::cout << "Available commands:\n";  
 std::cout << std::setw(32) << std::setfill('-') << '\n';  
 std::cout << "h: Help\n";  
 std::cout << std::setw(32) << std::setfill('-') << '\n';  
 std::cout << "1: Correct the text\n";  
 std::cout << "2: Filter & display words by letters or numbers presence\n";  
 std::cout << "3: Replace numbers to letters with the same index in the alphabet as number value\n";  
 std::cout << "4: Find substring in string using linear and KMP algorithms\n";  
 std::cout << "c: Change the sentence (or reread from the input.txt file)\n";  
 std::cout << std::setw(32) << std::setfill('-') << '\n';  
 std::cout << "0: Exit\n";  
 std::cout << std::setw(32) << std::setfill('-') << '\n';  
 std::cout << "NOTE: Script will skip all the useless input (e.g. extra symbols in action input)\n";  
 std::cout << std::setw(32) << std::setfill('-') << '\n';  
 std::cout << std::setfill(' ');  
 break;  
 }  
  
 // Unknown command error  
 default: std::cout << "RuntimeError: unknown command\n";  
 }  
 }  
  
 return 0;  
}